Пред-процесор

Пред-процесорът представлява програмна компонента, която преглежда и обработва всяка програма, преди компилатора. Той търси в програмата и изпълнява специални инструкции, които се наричат **директиви на пред-процесора**. Всяка директива на започва на нов ред със символът #.

включване на файл: # include <stdio.h> # include ”stdio.h”

```
/*Пример: функция за намиране на по-голямото от
две цели числа  записана във файл: max.exm в
текущата директория  */
int max (int first, int second){
   if (first >= second) return first;
   else return second;
}
```

```
/*тестова програма ползваща max.exm, записана във друг
файл - например с име prepr1.c  */
# include <stdio.h>      // в директориите за включване
# include ”max.exm”   //  в текущата директория
int main()
{   int i1,i2,mx;
    printf(”input 2 int numbers: ”);
    scanf (”%d%d”, &i1,&i2);
    mx = max(i1, i2);          //включената функция max()
    printf(”the max is:  %d”, mx);
   return 0;
}
```

макро-определения (макроси) –проста замяна на низове: #define низ1 низ2

макро-име макро разширение

Пред-процесорът преглежда кода на програмата и замества навсякъде макро-имената със съответните макро-разширения

```
#define PI  3.1415926536  // макро-разширение - числова константа
#define PROMPT  "\n  input the radius:"  //макро-разширение - текстов низ
#define MSG "The surface is:  %f\n "
          //макро-разширение - текстов низ с продължение на следващия ред
#define ERR_MSG "must be\
  positive numeric value "
#include <stdio.h>
#include <stdlib.h>
double area (double rad)                    //функция изчисляваща площта
{ return PI*rad*rad; }
int main() {
    double radius;
    char buf[30];                                   // буфер за съхраняване на входа
    do {                                            //повторение до правилно въвеждане
          printf (PROMPT);                          //без допълнителни кавички
          gets(buf);                                // въвеждане в буфера
          radius = atof(buf);                       //преобразуване в число
          if (radius >0) printf(MSG , area(radius));
          else puts(ERR_MSG);
    }while (radius <=0);
    return 0;
}
```

Забележки: • Макро-определението не завършва с точка и запетая. Ако бъде записан, то символът точка и запетая се включва в макро-разширението. • Макро-име включено в константен низ не се заменя с макро-разширение. Така ако в горния пример се зададе инструкцията: printf("PROMPT"); пред-процесорът няма да замени PROMPT с макро-разширението. • За да се различават от имената на променливите, обикновено макро-имената се задават с главни букви. •Макро-определението може да се включва на произволно място в програмата. Действието му се разпростира във файла след мястото на дефинирането му. Действието на макроса може да бъде отменено с директивата към пред-процесора #undef. Едно и също макро-име може да бъде дефинирано многократно. Ако едно и също макро-име е дефинирано многократно новата дефиниция отменя действието на последното въведено макро-определение:

```
#define BIG 10          //BIG е дефиниран като 10
#undef BIG              //BIG не е дефиниран
#define BIG 1000        //BIG е дефиниран като 1000
#define BIG 40          //BIG е дефиниран като 40
#undef BIG              //BIG е дефиниран като 1000
```

Трябва дебело да се подчертае, че използването на тези възможности силно намалява четливостта на програмата.

Макро-определения с аргументи. В макро-имената могат да участвуват аргументи заградени в скоби. Тези аргументи се включват и в съответстващото макро-разширение. Пример:

```
#define PI  3.1415926536
#define PROMPT  "\n  input the radius:"
#define MSG "The surface is:  %f\n "
#define ERR_MSG "must be\
  positive numeric value "
#include <stdio.h>
#include <stdlib.h>
#define AREA(rad) (PI*rad*rad)
int main() {
    double radius;
    char buf[30];
    do {
        printf (PROMPT);
        gets(buf);
        radius = atof(buf);
        if (radius >0) printf(MSG , AREA(radius));
        else puts(ERR_MSG);
    }while (radius <=0);
    return 0;
}
```

Всеки път, когато срещне комбинацията "AREA" следвана от фактически аргумент заграден в скоби, пред-процесорът я заменя с макро-разширението, в което формалният аргумент е заменен с фактическия. **Забележки:** •Правилото, че не може да има празни интервали в макро-имената важи и за макросите с аргументи. Ако в горния пример се остави празен интервал между AREA и (rad), то (rad) ще стане част от макро-разширението (вътре в скобите може да има празни интервали).
•Възможно е дефинирането на макро-имена с повече аргументи. Например:
#define EXMPL(x,y) (x*x + y) •Затварянето на макро-разширението в скоби не е задължително, но е препоръчително. Тъй като не е възможно да се предвиди в каква част на програмата ще се използува макро-разширението, то ограждането му в скоби ще осигури правилното му изпълнение независимо от приоритета на околните операции.
Разлика между функции и макроси. AREA(++radius)

Условна компилация. Директивите към пред-процесора **#ifdef**, и #**ifndef** позволяват да се управлява компилацията на програмата. Те имат следния синтаксис:

```
#ifdef <макро-име>
      // инструкции - компилират се при
      //дефинирано макро-името
#endif
```

или

```
#ifdef <макро-име>
      // инструкции, - компилират се,
      //при дефинирано макро-име
#else
      //инструкции - компилират се в
      //противен случай
#endif
```

```
#ifndef <макро-име>
      // инструкции - компилират се при
      //недефинирано макро-име
#endif
```

```
#ifndef <макро-име>
      // инструкции - компилират се при
      //недефинирано макро-име
#else
      //инструкции - компилират се в
      //противен случай
#endif
```

Действието и на двете директиви продължава до срещане на директивата *#endif*. Те могат да се включват една в друга.
Използването на тези директиви, може да направи програмата по-гъвкава и мобилна. Те се използват особено често в заглавни файлове. за избягване на многократно дефиниране на променливи и функции. Това позволява във всеки заглавен файл да се включват безопасно всички необходими дефиниции и други заглавни файлове.